Energy-aware Self-Adaptation for Application Execution on Heterogeneous Parallel Architectures

Richard Kavanagh, Karim Djemame, Jorge Ejarque, Rosa M. Badia and David Garcia-Perez

Abstract—Hardware in High Performance Computing environments in recent years have increasingly become more heterogeneous in order to improve computational performance. An additional aspect of such systems is the management of power and energy consumption. The increase in heterogeneity requires middleware and programming model abstractions to eliminate additional complexities that it brings, while also offering opportunities such as improved power management. In this paper we explore application level self-adaptation including aspects such as automated configuration and deployment of applications to different heterogeneous infrastructure and for their redeployment. This therefore not only mitigates complexities associated with heterogeneous devices but aims to take advantage of the heterogeneity. The overall result of this paper is a self-adaptive framework that manages application Quality of Service (QoS) at runtime, which includes the automatic migration of applications between different accelerated infrastructures. Discussion covers when this migration is appropriate and quantifies the likely benefits.

Index Terms—Self-adaptation, energy modelling, programming model, heterogeneous hardware architectures, application deployment.

1 Introduction

Advances in distributed computing research have in recent years resulted in considerable commercial interest in utilising heterogeneous hardware architectures (e.g. Central Processing Units (CPUs), Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs)), both with the intent of improving performance and reducing overall power and energy consumption. Solving issues with energy consumption is seen as a means of obtaining exascale performance [1]. Heterogeneous hardware's prevalence in HPC systems has led to significant complexities in scheduling work within the data centers. This increased heterogeneity has given rise to the need for abstractions that simplify the use of such infrastructures, ensuring that the maximum benefit may be obtained.

Usage of large scale systems with heterogeneous resources and multiple application implementations gives rise for the need to maintain Quality of Service (QoS). This is a complex task, given the multiple alternative applications implementations and the deployment options given the heterogeneous hardware. Each alternative application to hardware mapping, obtains different performance, power and energy characteristics, which need to be understood within the context of a large scale HPC system of competing application deployments.

Added to this complexity computer systems have faced significant power consumption challenges over the past 20 years. The dual challenge of both power and performance has in recent years shifted from the devices and circuits level, to their current position as first-order constraints for system architects and software developers. A common theme is the need for low-power computing systems that are fully interconnected, self-aware, context-aware and self-optimising within application boundaries [2].

The need for the maintenance of QoS within complex systems gives rise for the need for self-adaptation. Self Adaptive Systems have seen a significant level of interest in different research areas like autonomic computing and pervasive computing [3]. They provide self-management properties and exhibit system properties such as self-awareness to achieve adaptation. They are capable of monitoring their resources, state and behaviour.

Thus, QoS, power saving, performance and fast computational speed are key requirements in the development of applications. In this paper we address these issues with a energy-aware self-adaptive framework for heterogeneous parallel architectures. The main contributions of this paper are:

- a framework for managing heterogeneous hardware and optimising application deployments through self-adaptation
- a programming model that performs initial optimisation and task generation
- a event driven self-adaptation manager for runtime

- *K. Djemame and R. Kavanagh are with the School of Computing, University of Leeds, UK, LS2 9JT. E-mail: {K.Djemame,R.E.Kavanagh}@leeds.ac.uk*
- *J. Ejarque is with Barcelona Supercomputing Center (BSC), Spain. Email: {jorge.ejarque}@bsc.es*
- *Rosa M. Badia is with BSC and Artificial Intelligence Research Institute - Spanish National Research Council (IIIA-CSIC), Spain Email: {rosa.m.badia}@bsc.es*
- *D. Garcia-Perez is with Atos, Spain. Email: {david.garciaperez}@atos.net*

based adaptations
- recommendations and analysis on comparisons of deployment solutions on heterogeneous devices.

The remainder of the paper is organised as follows. In Section 2 we present the overall architecture that supports energy awareness and self-adaptation. Section 3 gives a detailed discussion of adaptation within the proposed framework. In Section 4 the experimental setup is discussed, followed by experimentation and evaluation in Section 5. The related work is then presented in Section 6 and Section 7 summarises the research and provides plans for future work.

2 Architecture

In order to meet the requirements for handling: power, QoS and hardware heterogeneity management we propose an architecture shown in Figure 1. It includes the high-level interactions of all components, separated into three distinct layers and follows the standard application deployment model. Its aim is to control and abstract underlying heterogeneous hardware architectures, configurations and software systems, while providing tools to optimize various dimensions of software design and operations (energy efficiency, performance, data movement and location, cost, time-criticality, security, dependability on target architectures). Next, details on the interactions of the architectural components are discussed.

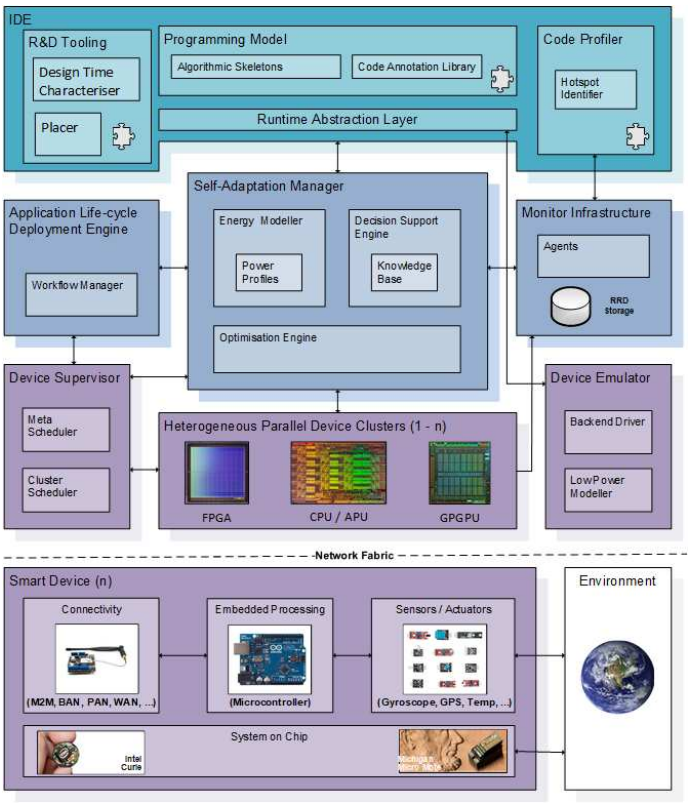


Fig. 1: Architecture

2.1 Layer 1 - IDE

The first block, Integrated Development Environment (IDE) facilitates the modelling, design and construction of applications, in order to aid the evaluation of power consumption. A number of IDE plug-ins are provided as a means for developers to interact with components within this layer. These are implemented as plug-ins which can be installed in Eclipse. Lastly, this layer enables architecture agnostic deployment of the constructed applications, while also maintaining low power consumption awareness. The components in this block are:

Requirements and Design Tooling: guides the development and configuration of applications determining what can be targeted in terms of both Quality of Service (QoS) and power consumption when exploiting heterogeneous hardware devices;

Programming model (PM): supports developers when coding their applications sequentially by letting them annotate their programs in such a way that the PM runtime can then execute them in parallel on heterogeneous parallel architectures being aware of the power consumption. The PM is implemented as an hierarchical combination of the COMPSs [4] and OmpSs [5] task-based programming models, where COMPSs manages the application in the distributed platform and OmpSs inside each compute node.

Code Profiler: This achieves power reduction through the software development process by providing developers the ability to directly understand the energy footprint of the code they write.

2.2 Layer 2 - Middleware

The second block handles the placement of an application considering energy models on target heterogeneous parallel architectures. Its tools are able to assess and predict performance and energy consumption of an application. Application level monitoring is also accommodated, in addition to support of self-adaptation for the purpose of making decisions using application level objectives. The components in this block are:

Application Life cycle Deployment Engine (ALDE): this component manages the lifecycle of an application deployed by the IDE. The ALDE introduces four entities for each application: Application, Executable, Deployment, and Execution Configuration. More details can be found in Section 3.2.

Monitor Infrastructure (MI): provides monitoring of the heterogeneous parallel devices (CPU, memory, network ...) along with historical statistics for device metrics. The monitoring of an application includes power, energy consumed and performance;

Self-Adaptation Manager (SAM): This component provides key functionality to manage the runtime based adaptation strategy applied to applications and Heterogeneous Parallel Devices (HPDs). This includes

aspects such as initiating redeployment to another HPD, restructuring a workflow task graph or dynamic recompilation. Furthermore, the component provides functionality to guide the deployment of an application to a specific HPD through predictive energy modelling capabilities and polices. Further details can be seen in Section 3.3.

2.3 Layer 3 - Heterogeneous Devices

The last block, addresses the heterogeneous parallel devices and their management. The application admission, allocation and management of HPDs are performed through the orchestration of a number of components. Power consumption is monitored, estimated and optimized using translated application level metrics. These metrics are gathered via a monitoring infrastructure and a number of software probes. At runtime HPDs will be continually monitored to give continuous feedback to the Self-Adaptation Manager. This ensures the architecture adapts to changes in the current environment including energy demand. The components in this block are:

Device Supervisor (DS): provides scheduling capabilities across devices during application deployment and operation. The component essentially realises abstract workload graphs, provided to it by the Application Life-cycle Deployment Engine, by mapping tasks to appropriate HPDs;

Device Emulator: provides the initial mapping of the application tasks onto the nodes/cores (at compile time). The mapping procedure is static and thus does not take into account any run-time constraints or run-time task mapping decisions.

3 Adaptation Framework

This section comprises of four main sub-sections, the first (Subsection 3.1) discusses the Programming Model, its role in adaptation at application construction. This is followed by Subsection 3.2 covering the application lifecycle management and how multiple deployment configurations can be constructed. After this runtime considerations are considered with the Self-Adaptation manager (Subsection 3.3) and how the adaptation can be guided by an energy model in Subsection 3.4.

3.1 Self-adaptation view at Programming Model

When developing an application with the PM, developers have to decompose applications into tasks. Candidates for becoming tasks are methods which are repeated in the application that can run independently from other tasks and require a certain computation for their execution. To define a task, developers have to annotate these methods indicating which arguments are input, output or input/output data. Based on this information, when executing an application, the PM runtime creates a Direct-Acyclic-Graph (DAG) of tasks dependencies where nodes are tasks to execute and arrows are data-dependencies between the tasks.

Developers can also define different task versions, so they can easily incorporate: sequential or multi-threaded versions, versions implemented as OpenCL or CUDA kernels, or different binaries calls which require different configurations. During the application execution, the runtime creates profiling information based on previous task version execution. This profile includes statistical information about the duration and power consumption of each task version. Based on this, the runtime select the most appropriate version for the available computing devices in terms of time or energy.

Moreover, the task dependency graph stores valuable information about what would be the computational load required by the application depending on the status of the application execution. The runtime can then obtain from the graph precise estimations of the application computational load without having to perform complex statistical analysis on monitored data collected from previous executions. Analysing the generated graph, the runtime can find the maximum achievable parallelism for a certain execution and compare it with the available resources. The runtime's analysis can detect the following situations:

1) When there are many ready tasks (dependency-free) which are pending to be executed, the application could run faster if it had more resources.
2) When there are resources that can run more tasks than the ready ones, we will waste resources because some of them will be idle or not running at the maximum performance.

To overcome this, we can configure the self-adaptation system with the following actuators:

- If the first situation occurs and there are unallocated resources, one of these resources has to be assigned to the task and then notify the application runtime. Once the runtime receives the notification, it starts the required processes in the node and executes pending tasks in the new resource. With this adaptation, the application execution is sped up. However, it has the side effect that the power consumption is increased due to increased resource usage. The energy consumption could also be increased depending on the parallelization overhead and the achieved execution time speed up.
- If the second situation occurs, the runtime reschedules the tasks to use the minimum number of resources at maximum performance and the idle resources can be released for application allocation. This adaptation reduces the power and increases the efficiency of the execution reducing the overall energy consumed by the application. In contrast, it may increase the execution time because reducing the resources we can also limit the application parallelism.

As previously mentioned, the actions described before

have some implications in power, energy and performance. This implies their invocation has to be taken in coordination with the other actuators, in order to ensure that the quality and constraints required by the user and the infrastructure are fulfilled. The PM runtime can detect if some of application constraints can be fulfilled or not. With the task dependency graph, the task profile and the resources description, it can estimate the application duration as well as the power and energy consumption. When a resource scale-up or scale-down is requested, it can estimate the effect of the new configuration. If the new resource configuration is not fulfilling the application quality requirements it can reject these scaling requests. On the other side, the runtime is not aware of the usage of the rest of the infrastructure. Therefore, additional self-adaptation management is required at the middleware layer in order to take other decisions forming a perspective that considers other applications as well as the infrastructure capabilities and constraints. Finally, these adaptation rules have been programmed for the PM Runtime, but may be ported to other task-based programming runtimes or elasticity engines which are aware of how many independent tasks are pending/running, which can dynamically reschedule resources.

3.2 ALDE Adaptation Enablement

An *Application* can be build by the ALDE and optimized for different heterogeneous architectures. Each application can have a series of *Executables* which is optimal for just CPU computation and another one that it is optimal for CPU+GPU and so on. The ALDE for a system that it is connected to can upload all the necessary executable files to that platform creating a *Deployment*. Finally, each deployment can have associated several *Execution Configurations*, which indicate which heterogeneous resources the Deployment needs.

Other hardware items can be added, such as FPGAs or Many-Core Processors. The configurations can then be ranked by the IDE tooling or benchmarking. This ranking helps the ALDE to take the best deployment decision and also the Self-Adaptation Manager in cases of reconfiguration. Once an execution request is received, this component must choose the infrastructure that is most suitable according to various criteria, e.g. energy constraints/goals that indicate the minimum energy efficiency that is required/desired for the deployment and operation of an application;

3.3 Runtime Self-Adaptation Management

The Self-Adaptation manager's (SAM) principle role is to manage application level adaptation at runtime, managing trade-offs between energy, power and performance within the framework. It is event driven, deciding for each event what adaptation to take and where it should be applied.

It works through a series of listeners that monitor the physical infrastructure, the jobs that are launched and the system clock for cron based events. This therefore requires interaction with the Monitoring infrastructure (for system and application based metrics) as well as the Device Supervisor and ALDE (for application based information). The listeners act as triggers generating events which through a sequence of rules then map to actuators that perform the required adaptation.

3.3.1 Event Generation

The first step in adaptation is a notification event which derives from the listeners. These events principally contain the following information:

Time: the timestamp of the event.

Value: a raw value representing how large the QoS breach is, i.e. the measured value of the violation.

Type of violation message: This is either a "violation" if the violation is detected, a "warning", or an informative indicator such as a event driven by the system clock has occurred.

Agreement Term: the metric to be monitored.

Guarantee Id: an identifier for each QoS constraint.

Operator: such as greater than, less than, equal.

Guaranteed Value: the value of the threshold.

Events (Host, Application and Clock) dependant upon their source must contain additional information. *Host* events additionally must contain the hostname, thus indicating the events' origin. *Application* based events must additionally record the application's name, a reference to the exact application instance and a reference to any application configuration information and specific firing rules as defined by the ALDE. *Clock* events, must hold a map for additional settings so that it can mimic either a host or a application based event. This allows clock events generated by previous rules firing to mimic host or application based events, facilitating features such as un-pausing an application. Events such as the following have the potential to trigger adaptation:

Boundary conditions on measurements: provide a reactive response to a QoS breaches, by setting constraints on application and host metrics.

Idle host detection: enables responses such as increasing application's resource utilisation or switching off underutilised resources. This includes the detecting of idle hosts with accelerators, enabling opportunities for redeploying and reconfiguring applications.

Host's failing/failed or in drain state: allows for self-healing, where applications can be reconfigured and redeployed on the remaining infrastructure. Draining hosts of existing jobs can be sped up.

Applications approaching deadline: This allows applications to be check-pointed close to completion in order to preserve work before eviction.

Application starting/completion: Useful to constrain execution to set times of the day, ensuring power hungry applications with low QoS requirements can be launched as required but run later.

Cron based events: create triggers based upon schedules, increasing flexibility, e.g. events such as unpausing jobs at a set time later on.

3.3.2 Adaptation Rules

On event notification the SAM works in *two phases*. The first considers the mapping between the type of notification and the actuators to use essentially the type of adaptation to make such as: redeploying an application to use accelerators or pausing an application. The second phase indicates the exact nature of this adaptation to take such as which application should be adapted and by how much. The first phase utilises adaptation rules that can be specified as a tuple of: ⟨Agreement Term, Comparator, Response Type, {Event Type}, {Lower Bound}, {Upper Bound}, {Parameters}⟩ which is utilised to determine the form of adaptation to take. Two examples of this are: ⟨IDLE_HOST+ACCELERATED,EQ, RESELECT_ACCELERATORS⟩ and ⟨IDLE_HOST+ACCELERATED, EQ, RESELECT_ACCELERATORS, WARNING, 0, 0, KILL_PREVIOUS=TRUE; application=gromacs⟩.

The latter optional values allow for stronger granularity ensuring the adaptation behaviour considers the scale of the notification event. This provides the flexibility to do things such as:

- Responding to warnings, in a different fashion to breaches or informative notifications.
- Observing the difference between the guaranteed value and the measured value and providing a stronger response if the deviation is further away (i.e. the lower bound and upper bound values).
- Parametrising the rules, so applications can further indicate how adaptation should occur e.g. clock based events such as ”it is out of working hours” can specify through parameters application information, thus allowing lower priority jobs to run.

Application and resource based events, derived from measurements utilise a threshold value, which determines how many events are required before a rule fires. This ensures that the temporary reporting of minor breaches can be ignored (e.g. if power consumption goes too high due to a short burst of CPU utilisation).

Once a rule has fired a recent history log prevents the same rule firing in rapid succession, thus avoiding over adaptation. After a short configurable amount of time (e.g. last minute), the rule can then be re-fired. The rules can optionally be set into a hierarchy so that if one rule cannot be applied additional rules that match the criteria may be used instead. This generates the prospect of either having fall back options for adaptation or an intensification of the adaptation response.

3.3.3 Scale of Adaptation

The second phase then decides upon the location of adaptation. This involves the usage of a decision engine that considers various parameters, such as the application configuration, QoS goals (e.g. save energy, cap power, reduce completion time) and the current environment to decide where the adaptation should take place.

The decision engines handle cases where information is lacking on what to adapt. This can include cases such as host based events and their transformation into actions applied to applications. This also applies to clock based events that may have originated from an application or host based event. The transformation process for hardware based events can be achieved: *randomly*, based upon the applications power consumption, or based upon the *last application instantiated* on the originating host. Clock events can be transformed into either host or application based events dependant upon the additional parameters attached to the event. They are transformed in order of precedence by: 1) Event data has *application* details attached. Originates from a call back event, where for example a pause action has specified when to resume. 2) Event data has *host* details attached. Similar to above but originates from a host based event instead. 3) The decision rule contains the *host* or *application* data.

3.3.4 Actuators

The actuation can cover many different forms of change. This can be at the level of applications and the workload for example: increasing/reducing an applications wall time, adjusting the wall time so its closer to the runtime, aiding scheduling and in particular backfilling, pausing and restarting applications or sets of applications, oversubscribe applications to resources, reselecting the accelerators an application uses, killing off an application, or it can be at host level such as starting and shutting down hosts and adjusting the cluster’s power cap.

One actuator stands out as being more complex than the rest, ”Reselect Accelerators” (see Algorithm 1). Its primary aim is to choose an application configuration that is better than the existing configuration. This may for example be switching from a single threaded CPU bound executable to a GPU accelerated version of the same application. This could be done to improve the accelerator utilisation. The algorithm firstly filters out configurations that are already running and thus have a head start upon any new instance starting. The second phase in the algorithm selects the new instance to launch. This works by ranking each configuration by either, power, energy or completion time. The best configuration is then selected so long as its power/energy or completion time is better than the existing running configuration. This ranking is performed based upon pilot jobs that are executed before hand. The process generating the ranking is as follows:

1) launch fixed workload pilot jobs: (either a single representative value, or uniform range, applied to each alternative configuration).
2) record the following information (application_id, job_number, completion time, energy consumption, configuration_id).

3) for a given job filter out configurations that cannot run and calculate the average power, energy consumption and completion time for a given application_id and configuration_id.
4) order by chosen ranking (completion time, power or energy.

The pilot jobs have a fixed workload (or a sequence of workloads that is repeated uniformly against each configuration), ensuring that each application configuration is compared fairly. This comparison gives a relative ranking between the configurations based upon the current hardware setup. It is considered that each application configuration has a relative affinity to each of the available resources on the testbed, therefore if the pilot jobs are repeated several times the likely improvement between configurations is going to be realised. An example of this affinity is where CUDA compiled applications must be launched upon a GPU enabled node, thus constraining a configuration to launch on a subset of the total amount of nodes available, which narrows the likely range of possible power, energy and completion time values for the fixed workload on the testbed. This process enables the ratio of improvement between the configurations to be determined. This includes aspects such as the likely energy consumption and average power consumption for running a pilot job or job (by relative ratio between configurations), which reflects complex aspects such as which resources a particular job was submitted to.

Algorithm 1 Reselection of Accelerators

```
procedure RESELECTACCELERATORS(String appName, String deploymentId, boolean killPreviousApp, RankCriteria rankBy)
    AppConfig currentConfig ← getCurrentConfigurationInUse(appName, deploymentId)
    AppDefintion appDef ← getApplicationDefintion(currentConfig)
    AppConfig validConfigs[] ← getValidConfigurations(appDef)  ▷ check resources availability and executables are compiled
    validConfigs[] ← removeAlreadyRunningConfigurations( validConfigs[])
    AppConfig selectedConfig ← selectConfig(validConfigs[], appDef, currentConfig, rankBy)
    startAndStopNewAndOldJobs(selectedConfig, appDef, currentConfiguration)
end procedure

procedure SELECTCONFIG(AppConfig validConfigs[], AppDefintion appDef, AppConfig currentConfig, RankCriteria rankBy)
    sort(validConfigs[],rankBy)
    if first(validConfigs[]).isRunning() then
        return null
    end if
    if isRankBetter(first(validConfigs[]),currentConfig) then
        return first(validConfigs[])
    end if
    return null  ▷ no better solution so return
end procedure
```

3.4 Energy Modelling

Adaptation inside the framework requires guidance, one such aspect regards application power and overall energy consumption. The self-adaptation manager needs to know the likely consequences of its actions in regards to aspects such as the power and energy consumed by an application or physical host. Application power consumption cannot directly be measured and is synthetic in nature, based upon attributing power consumption to an application dependant upon workload. Adaptation of applications based upon power consumption therefore requires a model to attribute this power.

The energy modeller (EM) [6] considers the major power consumers such as CPUs and other accelerators. In order to do this it has various models that may be used to attribute power consumption to an application. Two models have been specifically designed for physical hosts with accelerators, namely the `CpuAndAcceleratorEnergyPredictor` that utilises neural networks to apply a fit to the available calibration data and the `CpuAndBiModalAcceleratorEnergyPredictor` that determines power usage of an accelerator assuming an unutilised and heavily utilised state. The latter adaptor being useful in cases where the quality of calibration data is poor which causes calibration to fail, yet it still offers an estimate that can give a guide to any adaptation.

The `CpuAndAcceleratorEnergyPredictor` works as an additive model in which the CPU and the accelerator's utilisation is considered separately. The CPU is considered as polynomial fit of order 2, this has been chosen because if the model turns out to be linear then it will still provide a good fit, yet offers flexibility in cases where pure linearity does not hold [6].

The accelerator based calibration is written in such a way as to be as flexible as possible. It utilises a multilayer perceptron network with a single hidden layer. The amount of inputs is based upon the size of the calibration data gathered providing a single output. The size of the hidden layer is scaled to be $\sqrt{inputsize + outputsize}$, this ensures its size is sufficient but not so large as to cause it to be overly trained. The emphasis is therefore placed upon gathering training data of sufficient quality for the network to train correctly, ensuring that the parameters chosen have sufficiently strong influence on the power consumption.

The second predictor `CpuAndBiModalAcceleratorEnergyPredictor` also works in an additive fashion, with two sub models, one for the CPU and another for the accelerators in use. The CPU submodel uses the same polynomial model as the `CpuAndAcceleratorEnergyPredictor` model. This accelerator model performs clustering assuming two distinct states, one at the higher end of usage while the accelerator is active and another assuming the accelerator is idle. This model acts as an approximation for situations when the accelerator is only partially observable,

has limited training data, or when the training data for accelerator utilisation does not correlate well to power consumption. An example of limited observability and correlation is with Nvidia GPUs, whereby the clock frequency of a stream multiprocessor (SM) may be used as a substitute for utilisation. This due to Nvidia-smi's utilisation value reporting the percentage of time in a given interval where at least one SM is active [7], which has limited direct correlation to power consumption. Alternative routes such as application profiling which provide performance counters for each application remain impractical, given overheads and requirements to attach to every application running.

4 Experimental Design

To evaluate the feasibility of the adaptation features as outlined in section 3, the following section presents the experimental design to test the performance of three sample use cases that are utilised to show self-adaptation within various different contexts. Experiments are designed in the context of the energy efficient HPC framework presented in Section 2 and implemented by the TANGO project [2].

Objectives - The objective of the experimentation is to ascertain if the self-adaptation when monitoring applications in operation achieves dynamic energy management in each layer of the software stack. In particular, the first experiment examines the PM and the performance of variants of the same application. This is then advanced upon examining the conditions required at runtime to switch between application implementations and still save energy. Finally, the generalisability of the framework is explored in generating application level power capping.

Use Cases - The first of the three use cases focuses on self-adaptation features applicable when the whole architecture is utilised. The latter two use cases focus on runtime based self-adaptation features of the middleware which can be achieved for all type of applications.

In the first case an application is run where its parallelism and load depend on its input parameters. It allows the PM runtime to detect the application parallelism and adapt the number of resources used by the application to the application load and their implication in terms of performance and energy consumption. In the second case we see the prospect of hardware becoming available, this might be caused by the completion of another job for example. This presents the opportunity to trigger adaptation and redeploy an application and may include changing accelerators used. In scenario 3 time based rules for each application will ensure an application is paused during a "busy time of the day" and then resumed later, thus smoothing power consumption.

Testbed - The experiments are performed on a cluster using a subset of a bullx blade system. The testbed is composed of the following heterogeneous hardware resources: 4 bullx 515 nodes equipped with: 2 Intel Xeon E5-2470 (Sandy Bridge) at 2.3GHz, 12 X 16GB DDR3-1600 ECC SDRAM and 2 X 256GB SATA3 SSDs. Additionally two nodes ns50-51 with 2 Nvidia Kepler K20X GPUs each and nodes ns52-53 with 2 Intel Xeon Phi 5100 series (rev 11) KNC each. In addition to these nodes there are: 3 bullx B520 double compute blades (ns55-57), each equipped with: 2 Intel Xeon E5-2690 v3 (Haswell) at 2.6GHz with 16 X 16GB DDR4-RDIMM 2133DDR and 2 X 256GB SATA3 SSDs.

TABLE 1: Use Case Scenarios

Case	1) Programming Model Runtime Self-Adaptation	2) Hardware Becomes Available	3) Power Smoothing
Why	Application quality can be improved.	QoS improvement	Timed trigger event (based on workload)
Where	Applications		
What	Scale-up/down application resources	Migrate app/re-configure app	Pause application
How	Request more application resources	Redeploy application	Pause application, causing lower power consumption
Objective	Application resource usage optimization	Faster completion time and potential lower total energy	Smooth overall power consumption by delaying unimportant jobs

```
@constraint(ComputingUnits=24, ProcessorType='CPU')
@task(tpr=FILE_IN, trrh=FILE_OUT, gro=FILE_OUT)
def mdrun_pc(tpr, trr, gro, props):
    mdrun.Mdrun(tpr, trr, gro, props).launch()

@implement(source_class="gromacs", method="mdrun_pc")
@constraint(processors=[
        {'Type':'CPU', 'ComputingUnits':8},
        {'Type':'GPU', 'ComputingUnits':1}])
@task(tpr=FILE_IN, trr=FILE_OUT, gro=FILE_OUT)
def mdrun_pc_gpu(tpr, trr, gro, props):
    mdrun.Mdrun(tpr, trr, gro, props).launch()
```

Fig. 2: Definition of tasks with using the python binding of COMPSs

Applications - The experiments utilise Gromacs (http://www.gromacs.org/), an open source and widely utilised molecular dynamics simulation package. It is used to generate load within the testbed although any HPC application would have been suitable. Additionally Gromacs provides a realistic application that can be compiled into various alternative implementations such as Message Passing Interface (MPI) and CUDA. The first experiment utilises Pymdsetup [8] which is an application implemented on top of the Gromacs framework, which facilitates the setup and the execution of systems for molecular dynamics simulations.

From a system defined in a configuration file, Pymdsetup composes an algorithm which automatically generates the required information for different protein mutations; for each of these mutations it performs different

molecular dynamic simulations and evaluations using Gromacs. Originally, Pymdsetup executes the generated algorithm sequentially. However, nothing precludes exploring the protein mutations in parallel. The PM has therefore been used to parallelize and distribute the execution of the different steps of the algorithm upon the heterogeneous computing nodes. The porting of the application has been performed as explained below, including a code snippet as depicted in Figure 2.

1) The python functions which performed the different steps of the algorithm have been annotated as coarse-grain tasks. It is done by adding a *@task* decorator before the method definition indicating the type and direction of the parameters (IN, OUT, or INOUT). In the code snippet, a task definition is shown indicating which parameters are files and their direction. The rest are considered objects with IN direction by default.
2) Different task implementations are defined for those tasks where there are versions for different computing devices such as GPUs. This is done by adding a *@implements* decorator before the task definition indicating the task which is defining a version. The code snippet shows how to indicate that *mdrun_pc_gpu* is a version of *mdrun_pc*.
3) For the different versions, constraints are set in order to schedule tasks properly. It is done by adding a *@constraints* decorator before the task definition. In the code snippet, we can see that the *mdrun_pc* task requires 24 CPU cores and the *mdrun_pc_gpu* task requires 8 CPU cores and 1 GPU.

From the application user perspective with the ported version, a molecular dynamic system for simulation is defined in the same way as in the original one. However, instead of executing the simulation sequentially, the runtime evaluates the data dependencies between the different task invocations creating a task dependency graph and executes in parallel those tasks which are free of dependencies. Figure 3 shows the task dependency graph generated by the runtime for a system where two protein mutations are evaluated. In the graph, we can see that an independent workflow of Gromacs invocations is created to evaluate each protein mutation, and at the end, the results per mutation are merged to produce a plot with the results. Note that the *gromacs.mdrun_pc* and *gromacs.mdrun_pc_cpt* tasks have a version for running in CPUs and GPUs as previously mentioned. Finally varying the number of mutations, leads to different parallel computational load of the molecular dynamic system simulation.

5 Evaluation

The following section discusses the performance of the self-adaptation presenting an analysis of the experimental results.

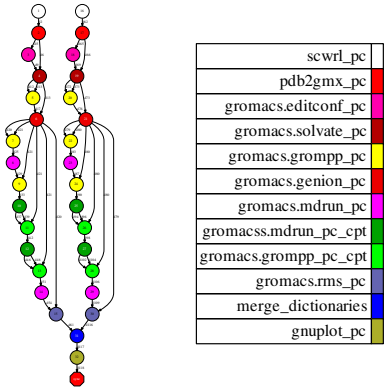


Fig. 3: Pymdsetup task dependency graph for two protein mutations

TABLE 2: Adaptation of task version to different resources

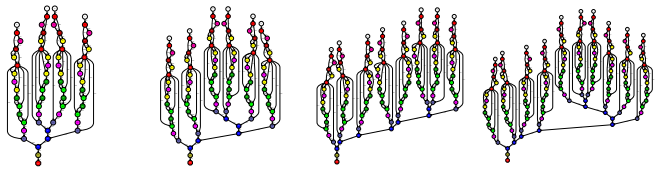
Node Type	Time(s)	Energy(KJ)	Executed version
B515 (CPUs/GPUs)	150.49	53.40	8 CPU + 1 GPU
B520 (CPUS)	203.41	63.58	24 CPU

5.1 Experiment 1

This experiment validates the self-adaptation features from the PM perspective. Pymdsetup application (presented in Section 4) is executed with different workloads in order to see how the PM runtime adapts the application execution taking into account the load and the available infrastructure. The experiment is split into three parts: the first one demonstrates how the runtime selects the appropriate version to achieve the maximum performance and energy consumption. The second part validates the adaptation of the resources according to the application load while the third one validates the adaptation of the resources according to the application quality requirements.

5.1.1 Task Version Selection Self-adaptation

The first PM self-adaptation feature selects the task version which is the most efficient for the available computing resources. To validate this feature, several task versions are defined for the *gromacs.mdrun_pc* and *gromacs.mdrun_pc_cpt* tasks which match with the capabilities of the different resource types. Then, Pymdsetup is executed to simulate a system with two protein mutations using different type of nodes: one execution uses the B515 nodes with 2 Kepler K20 GPUS; and another uses the B520 nodes. For both executions the execution time and energy consumption are measured. Table 2 shows that the runtime has selected the GPU version for the node B515 which has the best performance and energy consumption. In the case of the B520 nodes, it has selected the CPU task because the GPUs are not available. This adaptation did not require any change in the code.



(a) 4 mut. (b) 6 mut. (c) 8 mut. (d) 10 mut.

Fig. 4: Task-dependency graph for executions with different mutations (mut.)

TABLE 3: Resource adaptation according to quality requirements for 20 mutations simulation

Quality		Time	Energy	Used Nodes
Optimization	Constraints			
Time	none	427s	707KJ	5
Energy	none	1029s	670KJ	2
Time	max. ener. 700KJ	559s	693KJ	4
Energy	deadline 800s	751s	679KJ	3

5.1.2 Application Load Resource Self-adaptation

In the second part of this experiment, Pymdsetup is executed, simulating systems with a different number of mutations (4, 6, 8 and 10). How the runtime adapts the resources used by the application according to the execution load is then observed. Figure 4 shows the task dependency graphs generated by the different simulations and Figure 5 shows the parallel workload estimated by the runtime and the resources used for its execution. The parallel workload is the expected time to execute the dependency-free tasks in every evaluation interval. So, it is a combination of the number of tasks and its duration. This is compared by the capacity of the current infrastructure and the time which the Device Supervisor takes to allocate a new resource. According to this metric, the runtime decides to increase and decrease the number of resources. As seen in Figure 5, the runtime does not perform any adaptation for 4 mutations, because the parallel workload can be executed with the minimum configuration (2 nodes). In the case of 6, 8 and 10 mutations, the runtime scales-up the resources to 1, 2 and 3 resources to speed up the execution.

5.1.3 Quality Requirements Resource Self-adaptation

Finally, in the third part of the experiment, the application is executed to simulate a system with 20 mutations but with different quality requirements. In this case how the PM runtime adapts the application resources to fulfil the required quality is observed. Four executions are run. Two executions are configured for time optimization: one without constrains and another with an energy limit of 700KJ. The other two executions are configured for energy optimization: one without constraints and another with a deadline constraint. Table 3 shows the results of the experiment.

In the time optimization execution without constraints, the runtime's proposed solution is the same as the one in the previous section, to speed up the application to the maximum. Due to the large application load, the runtime scales-up the resources to the maximum (5 nodes). In contrast, when the execution is configured with energy optimization without constraints, the runtime decides not to scale-up the resources. This is because the energy estimation performed by the runtime with a larger number of resources is bigger than with the initial configuration. In the other two executions some constraints are added which limits or forces the runtime to achieve a certain level of parallelism. On one hand, in the case of time optimization with maximum energy constraint, the runtime tries to use the maximum parallelism, but limits the parallelism to 4 because the estimation with larger resources surpasses the constraint. On the other hand, in the case of energy optimization with deadline, the runtime is forced to use an extra resource because the estimated time is not fulfilling the deadline constraint.

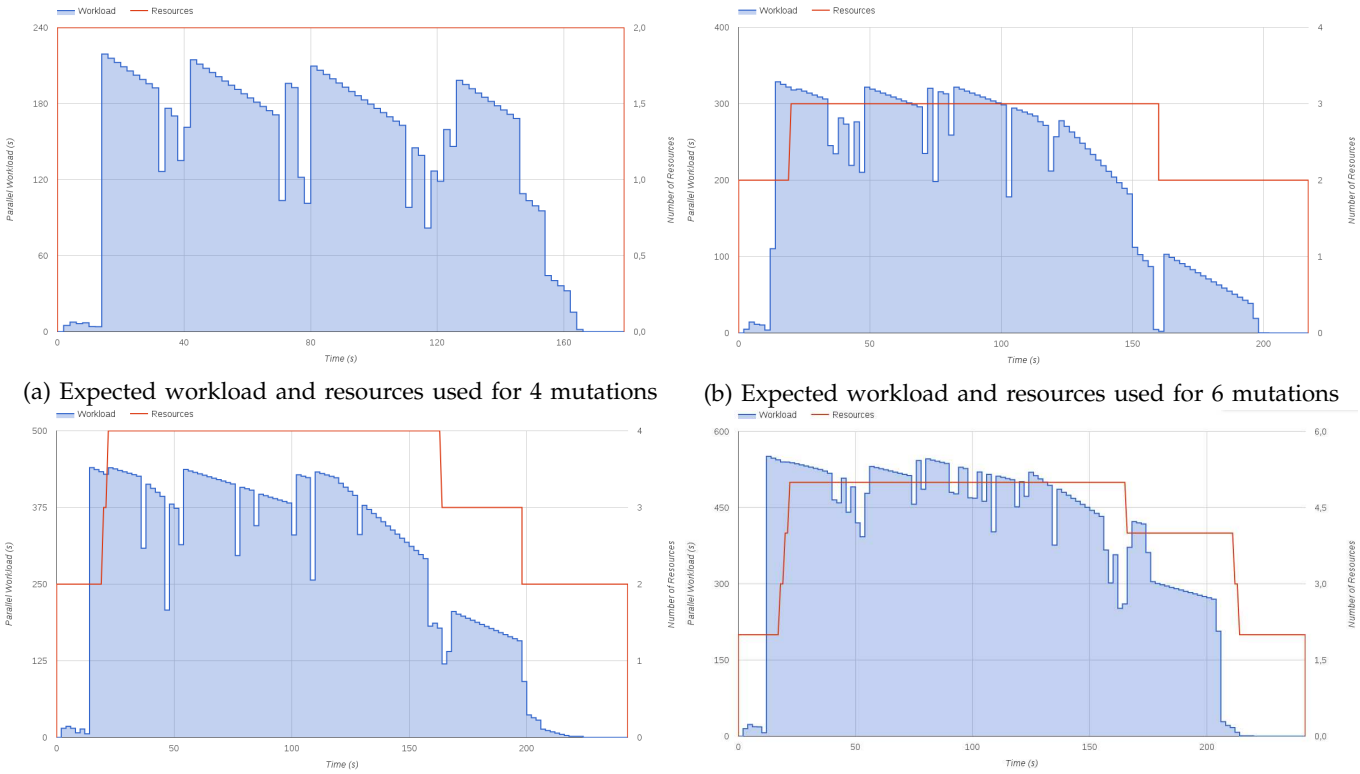
TABLE 4: Resource adaptation according to quality requirements for 10 mutations simulation

Quality		Time	Energy	Used Nodes
Optimization	Constraints			
Time	none	216s	342KJ	5
Energy	none	394s	317KJ	3
Time	max. ener. 350KJ	216s	342KJ	5
Energy	deadline 400s	393s	315KJ	3

In general, increasing the resources in the execution of a parallel application increases the energy consumption. This is due to large overheads such as transferring more files, etc. However, in some cases this is not the case, this part of the experiment is therefore repeated while halving the mutations and quality constraints, the results of which are shown in Table 4. For the time optimization without constraints, the solution reached by the runtime is the same as in the previous experiment, however for the energy optimization it has decided to scale-up up to 3 nodes. This is because the task execution is unbalanced with the minimum nodes. The runtime can run 4 mutation simulations in parallel, but in the last stage, there are only 2 mutations, so the resources are wasted. In the case with the resource adaptation, the runtime creates an extra resource to run the 2 pending simulations in parallel, thus solving the imbalance and reducing time and energy consumption. The same issue is experienced with the Time performance run with an energy limitation. Having 4 nodes we will have a unbalance load which is not happening with 5 nodes and therefore energy consumption is held below the threshold. Consequently this is detected by the runtime which decides to run the application with 5 nodes giving the same results as the time optimization without constraints.

5.2 Experiment 2

In this experiment the workflow covers a large proportion of the architectural components presented in Section 2, namely the SAM, ALDE, DS, EM and MI. They are used to illustrate the use of multiple application



(a) Expected workload and resources used for 4 mutations

(b) Expected workload and resources used for 6 mutations

(c) Expected workload and resources used for 8 mutations

(d) Expected workload and resources used for 10 mutations

Fig. 5: Resource self-adaptation

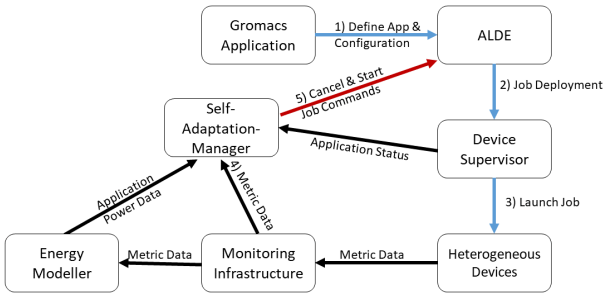


Fig. 6: Automatic Optimization of Applications

implementations of HPC applications in order to deploy, monitor and adapt an application so that the most efficient implementation is executed, given the resources that are available at the time of execution. The workflow is shown in Figure 6, is as follows: 1) The Gromacs application is defined in the ALDE and the configurations available 2) Job deployment, a CUDA implementation of Gromacs is started 3) the device supervisor (in our case SLURM) launches the job onto the infrastructure 4) The SAM receives an event indicating a host has become free with an accelerator 5) The SAM compares Gromacs implementations (in this case MPI vs CUDA but CUDA is already deployed) and re-launches the most efficient version of the application 6) The Gromacs application completes.

In the following experimentation two different versions of the Gromacs application are prepared, one using MPI (Using 16 threads) and the other CUDA. For each configuration, previous pilot runs are performed where execution time and energy consumption are measured for a given fixed workload. This presents a relative ranking of each configuration of the application upon the available hardware, given that each configuration will have an affinity towards a particular set of resources. A CUDA job for example must be launched on a subset of resources that have GPUs. Table 5 shows these initial measurements, where depending on the configuration, the Gromacs simulations can achieve different performance and energy consumption. The most efficient configuration in terms of time and energy for this execution is using the MPI implementation. The variance in execution time and energy consumption is low in this case, making it particularly suitable for determining speed up between configurations.

What can be seen is that the MPI application has the lowest energy consumption overall at 6,405.67J per run,

TABLE 5: Run of Pilot jobs to determine power, energy and completion time rankings

Name	Run Count	Total Energy (all runs)(J)	Average Energy (J)	Total Time (s)	Average Time Per Run (s)	Average power (W)
cuda	3	22,835	7,611.67	87	29	**262.47**
mpi	3	**19,217**	**6,405.67**	**67**	**22.33**	286.82

principally due to the lower runtime of 67s as compared to 87s. The average power consumption of the CUDA application is however less. This set of configurations therefore offers either: a lower power consumption that runs for longer and consumes more energy or a higher power consumption that runs for a shorter period of time and therefore uses less energy.

These results also give an indication of how quickly a replacement replica should start in order to consume less energy overall. This calculation assumes the workload is similar to that of the pilot jobs. To find the relative rank between jobs requires each application configuration to remain proportional in its energy usage to the other potential configurations. In the case of the pilot job executions (see: Table 5), an MPI implementation could be started in the first 4s of the CUDA instance's execution and still use less energy. This is derived from:

avg(cuda_run_energy) - avg(mpi_run_energy) = Δenergy_per_run
7611.67 - 6405.67 = 1206J
Δenergy_per_run / avg(mpi_run_power) = migration exploitation window size
1206 / 286.82 = 4.02s

Given the executions above are short lived the benefits of restarting jobs with different accelerators is limited, however if the workload is increased then the migration exploitation window size will also increase. Practically as this mechanism only provides the relative ranking between application configurations, without a priori knowledge of the runtime (to work out scale differences from the pilot jobs), then it is possible to use recent job submissions as guidance on current expected execution durations. The MPI and CUDA jobs both linearly scale (so far as tested), leaving the ratios between their durations and energy consumption comparable. However, due to gradient differences causing divergence (see Figure 7, CUDA increasingly uses more energy in comparison to the MPI implementation, therefore to save both energy and time, larger jobs should favour the MPI implementation. The migration exploitation window size scales linearly with the job size.

In addition applications using this calculation may be ranked to consider which ones have the most difference between the available configurations options. This therefore finds the application which is most likely to benefit from adaptation.

To illustrate the mechanism working the following rule: ⟨IDLE HOST ACCELERATED, EQ, RESELECT ACCELERATORS, WARNING, 0, 0, KILL PREVIOUS=TRUE; application=gromacs⟩ was created with the format: ⟨Agreement Term, Direction, Response Type, Event Type, Upper Bound, Lower Bound, Parameters⟩. It causes on detection of idle accelerated resources the SAM to consider if any Gromacs instance may be accelerated.

The reselection of accelerators mechanism compared the deployment options available and found an instance that has a lower amount of energy consumption was possible to execute. It therefore on receiving the IDLE HOST ACCELERATED event killed the previous instance and started a new job thus switching from the CUDA to the MPI implementation of Gromacs. The result of this is shown in Figure 8. The CUDA job runs on node ns51 and then is replaced by an MPI job on node ns55. The MPI job then completes at 27s, 2 seconds earlier than the CUDA job could have been expected to complete. Although this is a small benefit 6%, with longer running jobs the benefit is likely to be more substantial.

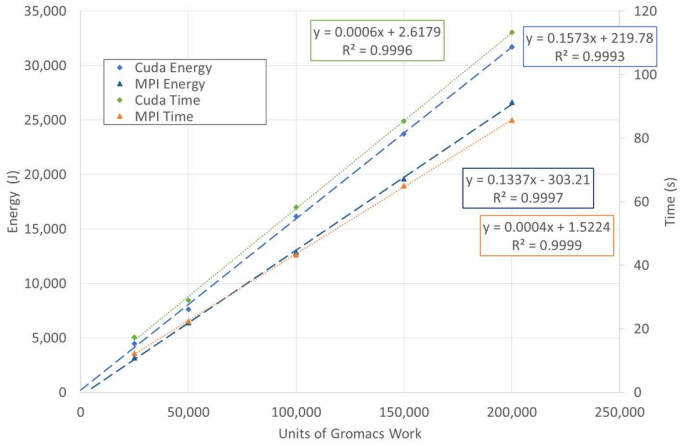


Fig. 7: CUDA vs MPI Job Workload Scaling

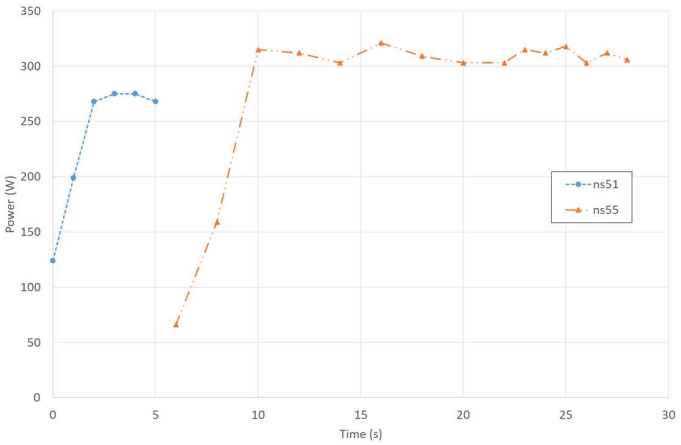


Fig. 8: MPI Job Launching and Cancelling CUDA Job

This job redeployment mechanism allows for multiple possibilities, when considering events that the SAM will act upon. For example job or node failure could cause the redeployment of a job to the most appropriate available resources, thus acting as a recovery mechanism. If ranking by power consumption it would aid power capping mechanisms by ensuring the mix of jobs running is most likely to have the least power consumption available. This mechanism also ensures accelerators where appropriate are more likely to be utilized.

In Figure 7 it seems counter intuitive that the CUDA implementation given the vast parallelism of the GPU does not perform better than the MPI implementation.

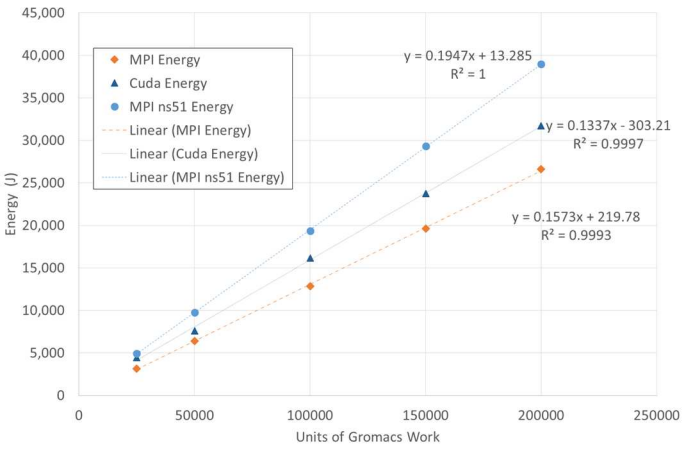


Fig. 9: MPI vs CUDA on Multiple Host Types

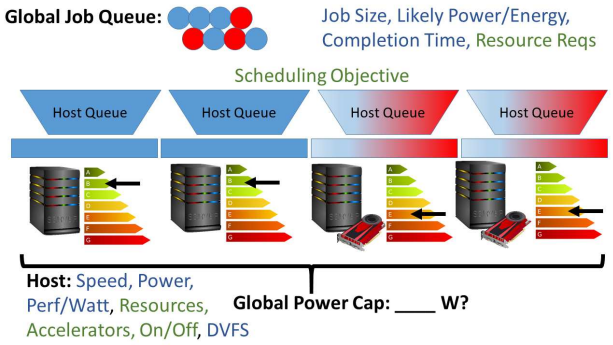


Fig. 10: Job Steering Considerations

This results in the transfer of the job from ns51 to ns55 in Figure 8. The difference is caused by the generational differences between the hosts leading to overall changes in efficiency. The explanation can be seen in Figure 9, which shows the energy consumption of a series of Gromacs jobs, both on ns51 (default choice for GPUs) and ns55 (default choice for MPI jobs), along with MPI implementation fixed so that it must use ns51. Here the graph for job completion time is omitted for purposes of clarity though it is similar to Figures 7 and 9.

The CUDA implementation is more efficient than the MPI implementation of ns51, however the MPI implementation can run on ns55 which is more efficient than ns51, therefore the MPI implementation is the best choice if the more efficient nodes are available.

5.2.1 Further Discussion

The strategy employed to determine the relative ranking of jobs has many considerations that must be made. These considerations (see Figure 10) are going to be split here into categories and discussed in turn. In the proposed framework, the application configurations bundle together resource requirements (CPU, Memory, GPU needs etc), application implementations (MPI, CUDA, other...) and QoS requirements. These constraining elements influence the direction to where jobs are directed and are indicated in green on Figure 10. In blue the QoS aspects are highlighted indicating how well an application configuration will perform once it arrives for processing. Figure 10 also shows a job queue with jobs in either a configuration for GPUs (red) or blue (general). This illustrates one constraint on applications, although this can be reconfigured to switch jobs between GPU only and general purpose. In terms of comparing various different configurations of an application two key aspects are identified: *temporal* and *power/energy*.

Job duration is important, simply the longer it runs the greater the power consumption. However ranking of jobs may be subject to changes in throughput for a given configuration, dependent upon the size of the workload (Figures 7 and 9). This can be checked with a series of jobs of different sizes and examining how the application scales under workload changes. If it scales in a predictable fashion, recently completed jobs can guide estimates for both completion time and the migration exploitation window size. Thus, this offers an estimate of the likely speed up/energy saving between configurations. One strategy of tackling throughput changes would be to make pilot jobs of a similar size to "typical" jobs thus mitigating scaling differences.

The second aspect relates to the power consumption of a job and its power consumption consistency (with obvious exception of the start and end of jobs). In the case of Gromacs (MPI) on ns55 the power consumption is consistent in the range of 303-320W with an average of 310W excluding earlier lower power consumption at the start of the applications execution. If it varies through time, does it act in phases with any particular recognisable fashion? and moreover does the average power consumption stay the same with job length?

Applications may be made up of several stages. These from the perspective of ranking configurations only need to be considered in cases where the average power consumption varies or the duration varies because of some underlying change in actions. Examples of this might include transferring data vs compute work, or different types of compute work on different accelerators/instruction sets. The steps may change size, altering average power consumption for example in cases where a large input file is needed. Data transfers may alternatively act as bounding behaviour on the power consumption, for example streaming might limit processor utilisation and hence power consumption as well.

An additional aspect is the heterogeneity of the hardware. An application's configuration is unlikely to specify specific hardware, but it may provide an affinity to a given subset of hardware or explicitly exclude some resources. An example of this is a CUDA job that needs a GPU. It may be for a given job configuration that the resources available to use are not very heterogeneous in terms of power and compute capacity (i.e. temporal/duration of job given the same workload). It is likely that this affinity of a given configuration towards different subsets of the infrastructure can be discovered by several runs of a given pilot job, thus providing an average case

for the speed up between configurations.

This strategy has its limitations and is dependent upon other workloads running at the same time, as well as the scheduler in use and size of the job to be rescheduled. The variance in time and energy consumption can be considered a measure of the reliability of the reselection method. The variance would be reduced if as part of the calculation the pilot run dataset could be reduced to records only considering the types of resources available or more specifically to the underlying resource that is going to be chosen.

5.3 Experiment 3

The third case helps smooth a data centre's power consumption by restricting certain applications power consumption at given times of the day, adding greater flexibility than merely a node level power constraint alone. The SAM can set a cluster's power cap at different times of the day or inc/dec-remented it based upon events. Application level power capping advances upon host level power capping in that it ensures choice upon when applications contribute to the power consumption.

In this experiment we start the SAM running with an application already running on the infrastructure, the power consumption of the application rises too high and the application is then paused. Later on the application is un-paused given a signal that it is now not running in normal working hours of the day, thus allowing it to complete its work, while ensuring the maximum power consumed is lowered during the prime hours of usage. There are two rules used to generate the pausing and resuming of jobs. Using a tuple with the format ⟨Agreement Term, Comparator, Response Type, Event Type, Parameters⟩ the following rules generate this behaviour. ⟨app_power:gromacs:*, GT, PAUSE APP, SLA_BREACH, application=gromacs;END_TIME= 14:49⟩ and ⟨out_of_hours, EQ, UNPAUSE SIMILAR APPS, WARNING, application=gromacs⟩.

The event for pausing an application was based upon power consumption at a given time of day. To ensure the power consumption would exceed the threshold a value of 50W was chosen. The rule for event generation follows the format: ⟨Unique Id, Agreement Term, Comparator, Event Type, Guarantee Value⟩ resulting in the following expression: ⟨1, app_power:gromacs:*, GT, SLA_BREACH, 50⟩. A cron rule for indicating out of hours (set for 14:50 for illustrative purposes only) was derived from the cron statement "0 50 14 1/1 * ? *" which triggers events with an agreement term of "out of hours". The rules outcome is for the application to be paused for a period of time, as shown in Figure 11. The result of pausing a selected application is that particularly power consuming applications can be moved to periods of time when the overall data centre power consumption is lower. This therefore allows the mechanism to smooth and select which applications contribute to the overall power consumption.

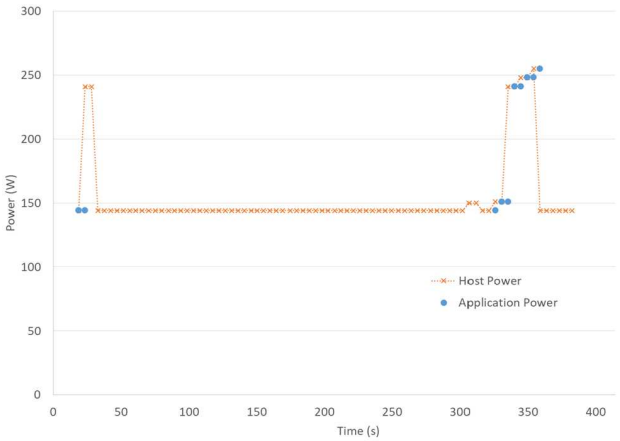


Fig. 11: Rule Based Application Pausing

6 Related Work

Research effort has targeted energy efficiency support at various stages of the application service lifecycle (construction, deployment, operation). On the other hand, heterogeneity has been recognized as a viable solution to improve performance and reduce power consumption at the same time [9]. This section reviews existing work and categorises it into self-adaptation, energy efficiency and heterogeneous computing environments.

Due to increasing systems complexity in recent years, there has been an increase trend towards self-adaptive systems (SASs) to address issue such as maintenance and configuration and maintenance of Quality of Service (QoS) in such complicated environments. Self-adaptation requires the answering of fundamental questions of "When to adapt?", "Why do we have to adapt?", "Where do we have to implement change?", "What kind of change is needed?", "Who has to perform the adaptation?" and "How is the adaptation performed?" [10].

Krupitzer et al. [3] presents a taxonomy of self-adaptive systems and their inspiration, while R. deLemos et al [11] identifies research challenges when developing, deploying and managing self-adaptive software systems. These challenges result from the dynamic nature of self-adaptation, which brings uncertainty. Adaptation can be considered to commonly follow a Monitor Analyses Plan and Execute (MAPE) approach. An extended architecture of the MAPE-K loop as a reference model for the design of self-adaptive systems is found in [12], assuming that the system has a central controller with a central MAPE-K loop. The proposal consists in continuously evaluating adaptation steps concerning their actual effect and adaptation mechanisms concerning their applicability and efficiency in the case of topology changes. An agent-based modelling approach for adaptation is presented in [13]. An Agent Verification Engine (AVE) which constructs agents to perceive, react, and adapt to runtime changes of a component-based system is proposed. These agents are based on the Belief-Desire-Intention (BDI) architecture, in which agents op-

erate in terms of motivation and beliefs. The work in [14] consolidates design knowledge of self-adaptive systems. To support software designers, the paper contributes with a set of formally specified MAPE-K templates that encode design expertise for a family of self-adaptive systems. The templates comprise: (1) behaviour specification templates for modelling the different components of a MAPE-K feedback loop (based on networks of timed automata), and (2) property specification templates that support verification of the correctness of the adaptation behaviours (based on timed computation tree logic).

The work presented in this paper focus upon low power and energy saving. Kong and Liu [15] group green-energy-aware into four distinct categories, namely: (1) Green-energy-aware workload scheduling. (2) Green-energy-aware Virtual Machine (VM) management. (3) Green-energy-aware energy capacity planning. (4) Interdisciplinary. The first three areas are focusing upon workloads temporal and spatial flexibility, VM scheduling for a similar effect and crafting workload profiles to desirable power consumption profiles. This paper focuses not only on workload placement but also upon the exact configuration of the application, which has multiple configurations available and various possibilities to use different hardware accelerators. The EcoScale project [16], [17] uses hardware performance monitors and models to project runtime and power consumption in heterogeneous environment using accelerators. The aim is to enable a runtime scheduler called the execution engine, to dynamically select and distribute software functions to either hardware acceleration or to software based execution. This selection is based upon workers in the vicinity and implementing worker queues [16].

Research effort has focused on the exploitation of hardware accelerators in cloud computing environments by addressing the challenge of programming such systems and making them easily accessible in a virtualized environment. One common approach is to propose methods to offload computations on heterogeneous hardware components. A solution for the efficient exploitation of specialised computing resources of a heterogeneous system is found in [18]. Other works have proposed heterogeneous architectures that combine high-performance and low-power servers in order to achieve better overall energy proportionality and energy efficiency [19]. The mapping problem between compute resources and application configurations is explored in [20], considering throughput in a cloud context. This is similar in idea to the work presented in this paper, though the context and approach differs regarding the heterogeneity that is being utilised. The ASCETiC project [21] holds similarities it focused upon energy efficiency and software adaptation but in the context of clouds. A highly scalable model for developing applications, exploiting hardware heterogeneity in cloud data centres while at the same time considering the aspect of energy efficiency is presented in [22]. In such model applications are expressed in the form of interconnected microservices which are automatically scheduled for execution on the most suitable heterogeneous computing elements. However, no evaluation of the model has been carried out.

The Legato project [23] identifies power as a key concern and while software-stack support for heterogeneity is relatively well developed for performance, is seen to remain an open question for power and energy-efficiency, which is an aspect that this paper contributes towards. StarPU [24] is one such early work that uses abstractions to allow workloads to be placed upon various different accelerator based platforms, selecting between various different accelerators, to improve computational performance and not energy. The Antarex project [25], [26], focuses on energy efficient systems and in particular on producing a tool chain that tunes code to run efficiently on heterogeneous infrastructures. The Manago project [27] is equally similar to work presented in this paper but with a particular focus on time-predictability with trade-offs with power and energy efficiency. Dutot et al. [28] advance upon power-capping and consider energy budgets with a principle focus on scheduling.

From a technology viewpoint hardware accelerators, such as GPGPUs and FPGAs still need the use of power reduction techniques such as Dynamic Voltage and Frequency Scaling (DVFS) and partial reconfiguration for FPGAs to keep power consumption under control [29]. Many approaches to adaptation and energy/power optimisation concentrate at the hardware level, such as utilising task scheduling coupled with GPU-specific DVFS and dynamic resource sleep (DRS) mechanisms, as a means to minimise the total energy consumption [30]. Our work in this paper compliments such hardware based strategies given the similarity of goals, yet utilises software based approaches to minimise power and conserve energy.

7 Conclusion

A key aspect for any future middleware system is to abstract away complexities of the heterogeneity and to support power and energy awareness through automatic reconfiguration at the level of the application. In this paper we have addressed these issues with a energy-aware self-adaptive framework for heterogeneous parallel architectures. This has included showing a self-adaptive approach for heterogeneous hardware, including more specific aspects such as job partitioning and allocation, automatic redeployment of applications and power capping. Discussion has focused on the criteria that would need to be considered as an element of the migration process. In a wider context of self-adaptation mechanisms have been demonstrated that can aid power capping behaviour and raise the intelligence of such an approach from the hardware to the middleware level. The self-adaptation manager alongside the other architectural components presented in this research are currently used in other frameworks e.g. Heterogeneous hardware and software alliance [31]. In future work

this will be extended to include check pointing based schemes as part of migration and utilising application models and programming model annotations to further rank application deployment alternatives during migration. This work on adaptation is also expected to be extended into other domains such as embedded systems.

ACKNOWLEDGMENTS

The authors would like to thank the European Commission for supporting this work under the Horizon 2020 Research and Innovation program under contract 687584 (TANGO project).

REFERENCES

[1] D. Stroobandt *et al.*, "EXTRA: Towards the exploitation of eXascale technology for reconfigurable architectures," in *2016 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, jun 2016, pp. 1–7.

[2] K. Djemame *et al.*, "Tango: Transparent heterogeneous hardware architecture deployment for energy gain in operation," in *Proceedings of the First Workshop on Program Transformation for Programmability in Heterogeneous Architectures*, Barcelona, Spain, March 2016, http://arxiv.org/pdf/1603.01407.

[3] C. Krupitzer *et al.*, "A survey on engineering approaches for self-adaptive systems," *Pervasive and Mobile Computing*, vol. 17, pp. 184 – 206, 2015.

[4] R. M. Badia *et al.*, "Comp superscalar, an interoperable programming framework," *SoftwareX*, vol. 3, pp. 32–36, 2015.

[5] A. Duran *et al.*, "Ompss: a proposal for programming heterogeneous multi-core architectures," *Parallel Processing Letters*, vol. 21, no. 02, pp. 173–193, 2011.

[6] R. Kavanagh, D. Armstrong, and K. Djemame, "Accuracy of Energy Model Calibration with IPMI," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. San Francisco, California: IEEE, jun 2016, pp. 648–655.

[7] NVidia Coporation, "NVML API Pages - For GPU Utilization," 2017. [Online]. Available: http://docs.nvidia.com/deploy/nvml-api/structnvmlUtilization__t.html#structnvmlUtilization__t.

[8] Barcelona Supercomputing Center and Institute for Research in Biomedicine, "Pymdsetup," 2017. [Online]. Available: https://github.com/bioexcel/pymdsetup

[9] S. P. Crago and J. P. Walters, "Heterogeneous cloud computing: The way forward," *Computer*, vol. 48, no. 1, pp. 59–61, Jan 2015.

[10] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, p. 14, 2009.

[11] R. de Lemos *et al.*, *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–32.

[12] V. Klos, T. Gothel, and S. Glesner, "Adaptive Knowledge Bases in Self-Adaptive System Design," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, aug 2015, pp. 472–478.

[13] K. Johnson, R. Sinha, R. Calinescu, and J. Ruan, "A Multi-agent Framework for Dependable Adaptation of Evolving System Architectures," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, aug 2015, pp. 159–166.

[14] D. G. D. L. Iglesia and D. Weyns, "Mape-k formal templates to rigorously design behaviors for self-adaptive systems," *ACM Transactions on Autonomous Adaptive Systems*, vol. 10, no. 3, pp. 15:1–15:31, Sep. 2015.

[15] F. Kong and X. Liu, "A Survey on Green-Energy-Aware Power Management for Datacenters," *ACM Computing Surveys*, vol. 47, no. 2, pp. 1–38, nov 2014.

[16] I. Mavroidis *et al.*, "Ecoscale: Reconfigurable computing and runtime system for future exascale systems," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 696–701.

[17] P. Harvey *et al.*, "A scalable runtime for the ecoscale heterogeneous exascale hardware platform," in *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers*, ser. ROSS '16. New York, NY, USA: ACM, 2016, pp. 7:1–7:8.

[18] G. Durelli, M. Coppola, K. Djafarian, G. Kornaros, A. Miele, M. Paolino, O. Pell, C. Plessl, M. D. Santambrogio, and C. Bolchini, "Save: Towards efficient resource management in heterogeneous system architectures," in *Reconfigurable Computing: Architectures, Tools, and Applications*, D. Goehringer, M. D. Santambrogio, J. M. P. Cardoso, and K. Bertels, Eds. Cham: Springer International Publishing, 2014, pp. 337–344.

[19] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. Moss, J. Mars, and L. Tang, "Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 246–258.

[20] M. Meredith and B. Urgaonkar, "On Exploiting Resource Diversity in the Public Cloud for Modeling Application Performance," in *The 9th International Conference on Cloud Computing, GRIDs, and Virtualization*. Athens, Greece: IARIA, 2017, pp. 66–72.

[21] K. Djemame, R. Bosch, R. Kavanagh, P. Alvarez, J. Ejarque, J. Guitart, and L. Blasi, "PaaS-IaaS Inter-Layer Adaptation in an Energy-Aware Cloud Environment," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 127–139, apr 2017.

[22] P. Ruiu, A. Scionti, J. Nider, and M. Rapoport, "Workload management for power efficiency in heterogeneous data centers," in *2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, July 2016, pp. 23–30.

[23] Legato Project, "LEGaTO Homepage," 2018. [Online]. Available: https://legato-project.eu/about

[24] C. Augonnet *et al.*, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 187–198, 2011.

[25] C. Silvano, G. Agosta, A. Bartolini, A. R. Beccari, L. Benini, J. Bispo, R. Cmar, J. M. P. Cardoso, C. Cavazzoni, J. Martinovič, G. Palermo, M. Palkovič, P. Pinto, E. Rohou, N. Sanna, and K. Slaninová, "Autotuning and adaptivity approach for energy efficient Exascale HPC systems: The ANTAREX approach," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 708–713.

[26] C. Silvano, G. Agosta, J. Barbosa, A. Bartolini, A. R. Beccari, L. Benini, J. Bispo, J. M. P. Cardoso, C. Cavazzoni, S. Cherubin, R. Cmar, D. Gadioli, C. Manelfi, J. Martinovič, R. Nobre, G. Palermo, M. Palkovič, P. Pinto, E. Rohou, N. Sanna, and K. Slaninová, "The ANTAREX tool flow for monitoring and autotuning energy efficient HPC systems," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2017, pp. 308–316.

[27] J. Flich, G. Agosta, P. Ampletzer, D. A. Alonso, C. Brandolese, E. Cappe, A. Cilardo, L. Dragić, A. Dray, A. Duspara, W. Fornaciari, G. Guillaume, Y. Hoornenborg, A. Iranfar, M. Kovač, S. Libutti, B. Maitre, J. M. Martínez, G. Massari, H. Mlinarić, E. Papastefanakis, T. Picornell, I. Piljić, A. Pupykina, F. Reghenzani, I. Staub, R. Tornero, M. Zapater, and D. Zoni, "MANGO: Exploring Manycore Architectures for Next-GeneratiOn HPC Systems," in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 478–485.

[28] P. F. Dutot, Y. Georgiou, D. Glesser, L. Lefevre, M. Poquet, and I. Rais, "Towards Energy Budget Control in HPC," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 381–390.

[29] J. Nunez-Yanez, "Energy efficient reconfigurable computing with adaptive voltage and logic scaling," *SIGARCH Comput. Archit. News*, vol. 42, no. 4, pp. 87–92, Dec. 2014.

[30] X. Mei, X. Chu, H. Liu, Y. W. Leung, and Z. Li, "Energy efficient real-time task scheduling on cpu-gpu hybrid clusters," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 1-4 May 2017, pp. 1–9.

[31] "Heterogeneous Hardware and Software Alliance Homepage," 2018. [Online]. Available: http://heterogeneityalliance.eu/